BUNDESREPUBLIK DEUTSCHLAND

PCT EP03/08081

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

REC'D 14 NOV 2003
WIPO PCT

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	102 40 022.9
Anmeldetag:	27. August 2002
Anmelder/Inhaber:	PACT XPP Technologies AG, München/DE
Bezeichnung:	Verfahren zum Debuggen rekonfigurierbarer Architekturen
IPC:	G 06 F 11/36

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 11. September 2003
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Stanschus

CERTIFIED COPY OF PRIORITY DOCUMENT

A 9161
06/00
EDV-L

Akte: PACT21d

Anmelder: Pact XPP Technologies AG
Muthmannstr. 1
80939 München

Vertreter: Patentanwalt
Claus Peter Pietruk
Heinrich-Lilienfein-Weg 5
D-76229 Karlsruhe
Vertreter-Nr. 321 605

Deutsche Patentanmeldung

Titel: Verfahren zum Debuggen rekonfigurierbarer Architekturen

Die vorliegende Erfindung befaßt sich mit Verfahren, für das Debuggen von Programmen auf konfigurierbaren Architekturen.
Unter einer rekonfigurierbaren Architektur, werden Bausteine (VPU) mit konfigurierbarer Funktion und/oder Vernetzung verstanden, insbesondere integrierte Bausteine mit einer Mehrzahl von ein- oder mehrdimensional angeordneten arithmetischen und/oder logischen und/oder analogen und/oder speichernden und/oder vernetzenden Baugruppen (im Folgenden PAEs genannt) und/oder kommunikativen/peripheren Baugruppen (IO), die direkt oder durch ein oder mehrere Bussystem(e) miteinander verbunden sind. PAEs sind beliebiger Ausgestaltung, Mischung und Hierarchie angeordnet. Diese Anordnung wird im Weiteren PAE-Array oder PA genannt.
Zur Gattung dieser Bausteine zählen systolische Arrays, neuronale Netze, Mehrprozessor Systeme, Prozessoren mit mehreren Rechenwerken und/oder logischen Zellen, Vernetzungs- und

Netzwerkbausteine wie z.B. Crossbar-Schalter, ebenso wie bekannte Bausteine der Gattung FPGA, DPGA, XPUTER, etc.. Hingewiesen wird insbesondere in diesem Zusammenhang auf die folgenden Schutzrechte desselben Anmelders: P 44 16 881.0-53, DE 197 81 412.3, DE 197 81 483.2, DE 196 54 846.2-53, DE 196 54 593.5-53, DE 197 04 044.6-53, DE 198 80 129.7, DE 198 61 088.2-53, DE 199 80 312.9, PCT/DE 00/01869, DE 100 36 627.9-33, DE 100 28 397.7, DE 101 10 530.4, DE 101 11 014.6, PCT/EP 00/10516, EP 01 102 674.7, PACT28, PACT26/US, PACT19, PACT20, PACT11. Diese sind hiermit zu Offenbarungszwecken vollumfänglich eingegliedert.

Weiterhin soll angemerkt werden, daß die Verfahren auch auf Gruppen von mehreren Bausteinen angewendet werden können.

Es werden mehrere Verfahren und Hardwareimplementierungen vorgestellt, die ein effizientes Debuggen von VPU-Systemen ermöglichen.

Die Aufgabe der vorliegenden Erfindung besteht darin, Neues für die gewerbliche Anwendung bereitzustellen.

Die Lösung der Aufgabe wird unabhängig beansprucht. Bevorzugte Ausführungsformen befinden sich in den Unteransprüchen.

1. Erfindungsgemäßes Verfahren

Das Debuggen erfolgt entweder durch die Verwendung eines entsprechend an eine VPU angeschlossenen Microcontrollers oder durch eine Ladelogik nach den Patenten PACT01, 02, 04, 05, 09, 10, 11, 13, 17, die durch Bezugnahme vollumfänglich eingegliedert sind.

Folgende grundlegenden Verfahren sollen dabei angewendet werden:

1.1 Erkennen einer Debug-Bedingung

1.1.1 Bedingung

Durch den Programmierer wird beispielsweise innerhalb des Debug-Tools eine oder mehrere Bedingungen festgelegt, die das Debugging starten (vgl Breakpoint nach dem Stand der Technik). Das Auftreten der Bedingungen wird zur Laufzeit in der VPU und/oder einem beliebigen mit der VPU datenaustauschenden Gerät festgestellt. Dies geschieht beispielsweise durch das Auftreten von bestimmten Datenwerten bei bestimmten Variablen und/oder bestimmten Triggerwerten bei bestimmten PAEs.

1.1.2 Vorbedingung

Im Optimalfall kann eine bestimmte Bedingung gemäß o.g. Definition bereits mehrere Takte vor dem Eintreffen der Debug-Bedingung durch den Programmierer festgelegt werden. Dadurch werden bestimmte Latenz-Probleme die im Folgenden diskutiert werden von vornherein ausgeschlossen.

Es sollen im folgenden zwei grundlegende Arten des Debuggings für VPUs diskutiert werden, wobei die jeweils bevorzugt anzuwendende Methode von der Wahl des Compilers abhängt:
Für Compiler, die Code auf Basis von instantiierten Modulen einer Hardwarebeschreibungssprache (oder ähnlichen Sprache) generieren, kann sich besonders die im Folgenden beschriebene Methode A eignen.
Für Compiler ähnlich PACT11, die komplexe Instruktionen erzeugen und nach einem VLIW-ähnlichen Verfahren generieren, eignet sich besonders die im Folgenden beschriebene Methode B. Grundsätzlich ist für den Betrieb als Prozessor oder Coprozessor Methode B die bevorzugt anzuwendende.

Es wurde erkannt, dass insbesondere die Verwendung beider Methoden A und B gemeinsam, zu den besten und transparentesten Debuggingergebnissen führt. Insbesondere kann je nach Tiefe des zu debuggenden Fehlers zunächst unter Zuhilfenahme der schnellen Debuggingmethode B debuggt werden und nach hinreichende Lokalisierung des Fehlers sodann per Methode A die Details in der "Tiefe" analysiert werden.

2. Methode A

2.1 Grundprinzip

Nach dem Auftreten einer (Vor)Bedingung wird die VPU angehalten. Danach werden die relevanten Debug-Informationen aus den PAEs an das Debug-Programm übertragen. Die relevanten Debug-Informationen wurden zuvor durch den Programmierer innerhalb des Debug-Programmes festgelegt. Nach Auslesen aller relevanter Debug-Informationen wird der nächste Takt ausgeführt und erneut die relevanten Debug-Informationen ausgelesen. Dies wiederholt sich solange, bis der Programmierer den Debug-Vorgang abbricht.

2.2 Unterstützung durch die Hardware

2.2.1 Auslesen der Register

Wesentlich für die Funktionsweise des Debuggers ist die Möglichkeit durch eine übergeordnete Einheit (im Folgenden DebugProzessor (DB) genannt), also beispielsweise eine CT oder Ladelogik, oder einen anderen extern angeschlossenen (Host)Prozessor, die internen Datenregister und/oder Statusregister und/oder Zustandsregister und ggf. je nach Implementierung weitere relevanten Register und/oder Signale aus den PAEs und/oder dem Netzwerk zurückzulesen (zusammengefaßt im Folgenden als Debuginformation bezeichnet). Eine derartige Möglichkeit ist z.B. mit der in PACT08/PCT geschaffenen Ver-

bindung zwischen der Ladelogik und dem Datenbus einer PAE (PACT08/PCT 0403, Fig.4) realisierbar.
Es soll ausdrücklich angemerkt sein, daß auch serielle Verfahren zum Auslesen der Register verwendet werden können. Beispielsweise kann JTAG gewählt werden und der DB über dieses Verfahren ggf. auch als externes seperates und evtl. auch marktübliches Gerät (z.B. Firma Hitex, Karlsruhe) angeschlossen sein.
Da bevorzugt auf sämtliche Register, oder zumindest eine erheblich Anzahl derer, durch den Debugger schreibend und/oder lesend zugegriffen werden kann, kann optional und bevorzugt ein wesentlicher Teil der (seriellen) Verkettung der Register zu Testzwecken (Scan-Chain) für die Produktionstests des Chips entfallen. Die Scan-Chain wird normalerweise dafür verwendet, um alle Register innerhalb eines Chips während der Produktionstests mit Testdaten vorladen zu können und/oder die Inhalte der Register zu Testzwecken wieder zurückzulesen. Das Vorladen und/oder Zurücklesen geschieht dabei typischerweise durch Testsysteme (z.B. SZ Testsysteme, Amerang) und/oder gemäß den in PACT09 beschriebenen Verfahren. Die Scan-Chain benötigt dazu einen zusätzlichen nicht unerheblichen Hardware- und Flächenaufwand für jedes Register. Dieser kann nunmehr, zumindest für die Register die debugbar sind, entfallen, wenn - wie erfindungsgemäß vorgeschlagen - die Produktionstestanlagen über geeignete Schnittstellen (z.B. parallel, seriell, JTAG, etc) Zugriff auf die Register erhalten.

2.2.2 Anhalten oder Verlangsamen des Taktes

Durch das Auftreten von Bedingung und/oder Vorbedingung kann der Takt entweder angehalten oder verlangsamt werden, um hinreichend Zeit zum Auslesen zur Verfügung zu stellen. Ausge-

löst wird dieser Debugbeginn entweder direkt durch eine PAE, die die (Vor)Bedingung(en) berechnete oder durch eine übergeordente Einheit (z.B. Ladelogik/CT, Hostprozessor) aufgrund beliebiger Aktionen, beispielsweise durch die Information, daß an einer PAE eine (Vor)Bedingung auftrat und/oder durch eine Aktion innerhalb des DebugProzessors und/oder durch ein beliebiges Programm und/oder eine beliebige externe/periphere Quelle. Zum Informieren stehen Triggermechanismen entsprechend PACT01, PACT02, PACT08, PACT10, PACT12, PACT17 zur Verfügung.

Sofern der Takt verlangsamt wird, muss durch den Debugprozessor innerhalb des verlangsamten Zyklus des Verarbeitungstaktes alle relevante Debug-Information aus den PAEs ausgelesen werden. Dazu ist es sinnvoll und bevorzugt, den Takt nur partiell zu verlangsamen, also den Arbeitstakt zu senken oder anzuhalten, aber den Takt für den Auslesemechanismus weiter fortzufahren. Desweiteren ist es sinnvoll und bevorzugt generell die Register zum Datenerhalt mit Takt zu versorgen.

Nach dem Anhalten des Taktes kann ein Single-Step Modus realisiert werden, d.h. der Debugprozessor hält den Verarbeitungstakt so lange an, bis er alle Debug-Information ausgelesen hat. Danach startet er den Verarbeitungstakt wieder für einen Zyklus und hält ihn erneut bis nach dem Auslesen aller relevanten Debug-Informationen an.
Der Auslesetakt und der Takt des Debug-Prozessors sind bevorzugt unabhängig vom Verarbeitungstakt der PAEs, sodaß die Datenverarbeitung von dem Debugging und insbesondere Auslesen der Debug-Information getrennt ist.
Hardwaretechnisch wird das Anhalten oder Verlangsamen des Taktes durch Methoden nach dem Stand der Technik, wie beispielsweise Gated-Clocks und/oder PLLs und/oder Teiler oder

andere Verfahren erreicht. Diese Mittel werden bevorzugt an geeigneten Stellen (Knoten) innerhalb des Clock-Trees eingefügt, sodass eine globale Taktsteuerungen der jeweils tieferliegenden Zweige realisierbar ist.

Besonders bevorzugt ist das Versenden einer Taktsteuer-Information von einer übergeordneten Einheit (z.B. Ladelogik/CT, Hostprozessor) an sämtliche PAEs. Dies kann bevorzugt über das Konfigurationsbussystem erfolgen. Die Taktsteuerinformation wird typischerweise gebroadcastet übertragen, d.h. alle PAEs erhalten dieselbe Information.

Beispielsweise können folgende Taktsteuerinformationen implementiert sein:

STOP: Der Arbeitstakt wird angehalten.

SLOW: Der Arbeitstakt wird verlangsamt.

STEP: Exakt ein Verarbeitungsschritt (Single Step Modus) wird ausgeführt und dann der Arbeitstakt wieder angehalten.

STEP(n): n Verarbeitungsschritte wird ausgeführt und dann der Arbeitstakt wieder angehalten.

GO: Der Arbeitstakt läuft normal weiter.

Es soll insbesondere erwähnt sein, dass das Verfahren der Taktabschaltung und/oder Verlangsamung ebenso eingesetzt werden kann, um den Stromverbrauch zu reduzieren. Sofern temporär keine Rechenleistung benötigt wird, kann ein "Sleepmode" beispielsweise durch das Abschalten des Arbeitstaktes (STOP) oder durch spezielle Befehle (SLEEP) realisiert werden. Wird nicht die volle Rechenleistung benötigt, kann der Takt durch die Verwendung von SLOW verlangsamt werden und/oder temporär durch STEP(n) ausgesetzt werden. Insoweit ist das Verfahren optional oder zusätzlich zu den in PACT15 beschriebenen Verfahren zur Reduzierung der Verlustleistung einsetzbar.

Ein Problem des Broadcastings von Taktsteuerinformationen ist die Übertragungszeit des Broadcastes durch das Array aus

PAEs. Die Übertragung kann bei höheren Taktfrequenzen nicht innerhalb eines Arbeitstaktes erfolgen. Es ist aber zwingend notwendig, dass sämtliche PAEs zur gleichen Zeit auf die Taktsteuerinformationen reagieren. Bevorzugt wird die Taktsteuerinformation daher über ein gepipelintes Bussystem ähnlich des in PACT17 beschriebenen CT-Bussystems übertragen. Weiterhin wird ein Zahlenwert (LATVAL) den Taktsteuerinformationen hinzugefügt der gleich oder größer der maximalen Länge der Pipeline des Bussystems ist. In jeder Pipelinestufe wird taktweise der Zahlenwert dekrementiert (Subtraktion von 1). Jede PAE die die Taktsteuerinformation erhält dekrementiert im Weiteren den Zahlenwert mit jedem Takt. Damit ist sichergestellt, dass der Zahlenwert in dem gepipelinten Bussystem und den PAEs die die Taktsteuerinformation bereits empfangen haben immer exakt gleich ist. Erreicht der Zahlenwert den Wert 0 ist sichergestellt dass alle PAEs die Taktsteuerinformation erhalten haben. Jetzt tritt die Taktsteuerinformation in Kraft und das Verhalten des Taktes wird entsprechend modifiziert.

Durch das beschriebene Verfahren entsteht eine weitere Latenzzeit (Latency). Diese kann beispielsweise durch die nachfolgend beschriebene Registerpipeline zusätzlich abgefangen werden oder, wie besonders bevorzugt, durch die Definition der (Vor)Bedingung abgefangen werden, indem die (Vor)Bedingung soweit vorgesetzt wird, dass die Latenzzeit bereits berücksichtigt ist.

Es soll besonders erwähnt werden, dass die Latenzzeit im Single-Step Modus vernachlässigbar ist, da sie nur bei der Abschaltung es Taktes (STOP) eine Rolle spielt. Da der Befehl STEP immer nur exakt einen Schritt ausführt kommt es während des Single-Step Betriebes durch keinerlei Verfälschung (Verzögerung) der Debugdaten durch die Latenzzeit.

2.2.3 Registerpipeline zum Ausgleichen der Latency

Bei höheren Betriebsfrequenzen kann es zu einer Latenzzeit zwischen dem Erkennen des Debugbeginns und dem Anhalten oder Verlangsamen des Taktes kommen. Diese Latenzzeit ist exakt vorherbestimmbar, da die Position der verzögernden Register in der VPU durch Hardware und/oder den zu debuggenden Algorithmus definiert und durch den Debugger daher exakt berechenbar ist.

Durch die Latenzzeit verschieben sich jedoch die dem Debug-Prozessor zur Verfügung gestellten Informationen derart, daß nicht mehr die korrekten Debuginformationen ausgelesen werden können. Bevorzugt wird dies durch ein entsprechendes festlegen der (Vor)Bedingung durch den Programmierer gelöst.

Optional kann durch das Einfügen eines mehrstufigen Registerpipeline, die die Debuginformation in jedem Takt um ein Register weiter überträgt, der Debugprozessor auf soviele Takte an Debuginformationen zurückgreifen, wie die Registerpipeline lang ist. Die Länge der Registerpipeline ist so auszulegen, dass sie der maximal zu erwartenden Latenz entspricht. Aufgrund der exakten Berechenbarkeit der Latenzzeit ist es nunmehr dem Debug-Programm möglich die zeitlich korrekte relevante Debug-Information aus der Registerpipeline auszulesen.

Ein auftretendes Problem bei der Verwendung von Registerpipelines ist, dass diese verhältnismäßig lang und damit, bezogen auf die für die Realisierung notwenige Siliziumfläche, teuer sind.

2.3 Sichtbare Debug-Informationen

Bei dieser Methode wird im Wesentlichen nach Auftreten der (Vor)Bedingung debugt, da erst danach in der Takt verlangsamt oder angehalten wird und das Auslesen der Debug-Information

erfolgt. Debug-Informationen die vor dem Auftreten der (Vor)Bedingung liegen sind daher zunächst nicht sichtbar.

Es ist jedoch, wenngleich auch unter Verlust an Arbeitsleistung, möglich eine VPU sofort vom Start einer Applikation an mit verlangsamten Takt oder Single-Step-Modus zu betreiben. Vom Start an werden die relevanten Debug-Informationen vom Debug-Prozessor ausgelesen.

3. Methode B

3.1 Grundprinzip

Relevanten Debug-Informationen aus den Speichereinheiten, die gemäß PACT01, 04, 13, 11, 18 die Anwendungsdaten und Zustände eines bestimmten Arbeitsschrittes beinhalten an das Debug-Programm übertragen. Diese Speichereinheiten, nachfolgend auch Arbeitsspeicher genannt, arbeiten im Maschinenmodell nach PACT01, 04, 11, 13, 18 quasi als Register zur Speicherung von Daten, die innerhalb eines Konfigurationszykluses im PA oder Teilen des PAs berechnet wurden. Es wird insbesondere auf die Patentanmeldung PACT11 verwiesen, in der die Verwendung der Speichereinheiten als Registers (REG) zur Relisierung eines Prozessormodells detailliert beschrieben ist. PACT11 ist zu Offenbarungszwecken vollumfänglich eingegliedert. Eine Speichereinheit besteht dabei aus einer beliebigen Anordung und Hierachie von unabhängigen und abhängigen Speichern. Es ist möglich gleichzeitig mehrere unterschiedliche Algorithmen auf dem PA auszuführen, die dann unterschiedliche Speicher verwenden.

Wesentlich für die Anwendung dieser Methode ist, dass Daten und/oder algorithmisch relevante Zustände in die den PAEs zugeordneten Speichereinheiten abgelegt sind. Wobei eine Speichereinheit jeweils zumindest derart dimensioniert ist, daß

alle relevanten Daten und/oder Zustände eines Zyklus gespeichert werden können; wobei die Länge eines Zyklus durch die Größe der Speichereinheit bestimmt sein kann und bevorzugt auch bestimmt ist (vgl. PACT04).

Unterschiedliche Daten und/oder Zustände werden derart in die Speichereinheiten gespeichert, daß diese eindeutig dem Algorithmus zugeordnet werden können. Dadurch kann der Debugger die relevanten Daten und/oder Zustände (Debug-Information) eindeutig identifizieren.

Die relevanten Debug-Informationen können - insbesondere auch vorab - durch den Programmierer innerhalb des Debug-Programmes festgelegt werden. Diese Debug-Informationen werden aus den Speichereinheiten ausgelesen. Dazu stehen unterschiedliche Methoden zur Verfügung, einige Möglichkeiten werden nachfolgend nähers beschrieben. Nach dem Auslesen aller relevanter Debug-Informationen wird der nächste Konfigurationszyklus ausgeführt und erneut die relevanten Debug-Informationen ausgelesen. Dies wiederholt sich solange, bis der Programmierer/Debugger den Debug-Vorgang abbricht.

Mit anderen Worten, werden die relevanten Daten und/oder Statusinformationen nicht taktweise sondern konfigurationsweise an den Debugger übertragen. Dies geschieht aus den Speichereinheiten, die mit den Registern einer CPU vergleichbar sind.

3.2 Unterstützung durch die Hardware

Wesentlich für die Funktionsweise des Debuggers ist die Möglichkeit durch die CT oder einen anderen extern angeschlossenen Prozessor (im Folgenden DebugProzessor (DB) genannt),

die, beispielsweise auch interne(n), Arbeitsspeicher der VPU zu lesen. Eine derartige Möglichkeit ist z.B. durch den Anschluß der CT an die Arbeitsspeicher zum Vorladen und Lesen der Daten und/oder durch die in PACT13 beschriebene Verfahren zum Herausschreiben der internen Speicher an externe Speicher gegeben. In einer mögliche Ausgestaltung kann auf Arbeitsspeicher durch verschieden Verfahren nach dem Stand der Technik (z.B. Shared Memory, Bank Switching) derart vom DebugProzessor zugegriffen werden, dass der Datenaustausch mit dem DB weitestgehend unabhängig von einer weiteren Datenverarbeitung in der VPU erfolgen kann.

In einer möglichen Ausgestaltung kann z.B. entsprechend Methode A durch eine oder mehrere der vorstehend beschriebenen Maßnahmen der Takt der VPU zum Auslesen der Speicher gegebenenfalls entweder entsprechend verlangsamt oder angehalten werden und/oder gegebenenfalls im Single-Step-Modus betrieben werden. Dabei kann je nach Implementierung der Arbeitsspeicher, z.B. bei Bank-Switch Verfahren auf einen besondere Eingriff in den Takt verzichtet werden.
Typischerweise geschieht das Anhalten oder Verlangsamen des Taktes nach Methode B und das Auslesen und/oder Kopieren und/oder Umschalten der Arbeitsspeicher nur, wenn ein Datenverarbeitungszyklus bzw. Konfigurationszyklus beendet ist.

Mit anderen Worten ist ein wesentlicher Vorteil von Methode B, dass keine besondere Unterstützung durch die Hardware erforderlich ist.
In einer möglichen Ausgestaltung muß ein DB lediglich Zugriff auf die Arbeitsspeicher besitzten.
In einer besonders zu bevorzugenden Ausgestaltung geschieht der Zugriff auf die Arbeitsspeicher durch eine geeignete Kon-

figuration der VPU, die dadurch die Arbeitsspeicher selbständig und ohne Modifikation ausliest und an einen DB überträgt.

3.3 Zugriff auf Debug-Information

In PACT01, PACT04, PACT11, PACT13 sind Datenverarbeitungsverfahren beschrieben, bei denen zyklisch eine Menge an Operationen auf einen rekonfigurierbaren Datenverarbeitungsbaustein abgebildet werden. In jedem Zyklus wird eine Mehrzahl von Daten berechnet, die von einer peripheren Quelle und/oder einen internen/externen Arbeitsspeicher stammen und an eine peripheren Quelle und/oder einen internen/externen Arbeitsspeicher geschrieben werden. Dabei können jeweils unterschiedliche und/oder vor allem mehrere unabhängige Arbeitsspeicher gleichzeitig verwendet werden.
Beispielsweise können in diesem Datenverarbeitungsverfahren die Arbeitsspeicher oder ein Teil der Arbeitsspeicher als Registersatz dienen.
Nach PACT11 und PACT13 werden dabei sämtliche Daten und Zustände die für die weitere Datenverarbeitung relevant sind in die Arbeitsspeicher abgelegt oder aus selbigen gelesen.
In einem bevorzugten Verfahren werden für die weitere Datenverarbeitung irrelevante Zustände nicht gespeichert.

Die Unterscheidung zwischen relevanten und irrelevanten Zuständen soll an folgendem Beispiel aufgezeigt werden, es soll zu Offenbarungszwecken insbesondere auf die Ausführungen in PACT11 verwiesen werden:
Die Zustandsinformation eines Vergleichs ist beispielsweise für die weitere Verarbeitung der Daten wesentlich, da dieser die auszuführenden Funktionen bestimmt.
Ein sequentielle Dividierer entsteht beispielsweise durch Abbildung eines Divisionsbefehles auf eine Hardware, die nur

die sequentielle Division unterstützt. Dadurch entsteht ein Zustand der den Rechenschritt innerhalb der Division kennzeichnet. Dieser Zustand ist irrelevant, da für den Algorithmus nur das Ergebnis (also die ausgeführte Division) erforderlich ist. In diesem Fall werden also lediglich das Ergebnis und die Zeitinformation (also die Verfügbarkeit) benötigt.
Die Zeitinformation ist beispielsweise in der VPU-Technologie nach PACT01, 02, 13 durch das RDY/ACK Handshake erhältlich. Hierzu ist jedoch besonders anzumerken, dass das Handshake ebenfalls keine relevanten Zustand darstellt, da es lediglich die Gültigkeit der Daten signalisiert, wodurch sich wiederum die verbleibende relevate Information auf die Existenz gültiger Daten reduziert.

In PACT11 wird eine Unterscheidung zwischen lokal und global relevanten Zuständen aufgezeigt:
lokal: Der Zustand ist nur innerhalb einer einzigen abgeschlossenen Konfiguration relevant. Daher muß der Zustand nicht zwingend gespeichert werden.
global: Die Zustandsinformation wird für mehrere Konfigurationen benötigt. Der Zustand muß gespeichert werden.

Es ist nunmehr denkbar, daß der Programmierer einen lokal relevanten Zustand debuggen will, der nicht in den Speichern abgelegt ist.
In diesem Fall kann die Applikation dahingehend modifiziert werden, daß eine Debug-Konfiguration entsteht (äquivalent zum Debug-Code von Prozessoren), die eine Modifikation zum "normalen" Code der Applikation derart aufweist, daß dieser Zustand zusätzlich in die Speichereinheit geschrieben und damit dem Debugger zur Verfügung gestellt wird. Dadurch entsteht eine Abweichung zwischen Debug-Code und tatsächlichen Code,

die zu einem unterschiedlichen Verhalten der Codes führen kann.

In einer daher besonders bevorzugten Ausführung wird keine Debug-Konfiguration verwendet. Vielmehr wird die zu debuggende Konfiguration derart terminiert, dass die für Debuggingzwecke zusätzlich erforderlichen Daten die Terminierung überdauern, d.h. weiterhin gültig in den entsprechenden Speicherstellen (REG) (z.B. Register, Zähler, Speicher) verbleiben.

Eine Konfiguration wird geladen, diese verbindet die REG in geeigneter Weise und definierter Reihenfolge mit einem oder mehreren globalen Speicher(n) auf die der DB Zugriff hat (z.B. Arbeitsspeicher).
Die Konfiguration kann beispielsweise Adressgeneratoren verwenden um auf den/die globalen Speicher zuzugreifen.
Die Konfiguration kann beispielsweise Adressgeneratoren verwenden um auf als Speicher ausgestaltete REG zuzugreifen.
Entsprechend der konfigurierten Verbindung zwischen den REG werden die Inhalte der REG in einer definierten Reihenfolge in den globalen Speicher geschrieben, wobei die jeweiligen Adressen von Adressgeneratoren vorgegeben werden. Der Adressgenerator generiert die Adressen für den/die globalen Speicher(n) derart, dass die beschriebenen Speicherbereiche (DEBUGINFO) der entfernten zu debuggenden Konfiguration eindeutig zugeordnet werden können.
Das Verfahren entspricht dem Kontext-Switch in PACT15 und PACT11 die zu Offenbarungszwecken vollumfänglich eingegliedert, sind.
Der DB kann nunmehr auf die Daten innerhalb eines ihm zugänglichen Speicherbereiches (DEBUGINFO) zugreifen.
Soll das Debugging mittels eines Single-Step Verfahrens durchgeführt werden, kann nach jedem single step einer zu de-

buggenden Konfiguration ein Kontext-Switch derart durchgeführt werden, dass sämtliche Daten erhalten bleiben und die zu debuggende Information aus den REG in einen Arbeitsspeicher geschrieben wird. Sodann wird wiederum unter Erhalt der Daten die zu debuggende Konfiguration wieder konfiguriert und für einen weiteren single step ausgeführt. Dies geschieht für jeden zu debuggenden single step der zu debuggenden Konfiguration.

3.4 Sichtbare Debug-Informationen

Ein Debuggen vor der (Vor)Bedingung kann vergleichsweise einfach und ohne große Performance-Verluste durchgeführt werden, da die benötigten Debug-Informationen in Arbeitsspeicher verfügbar sind. Durch das Übertragen der Arbeitsspeicher in andere Speicherbereiche, auf die der DB bevorzugt direkten Zugriff hat, kann die Debug-Information einfach sichergestellt werden. Eine noch schnellere Methode ist, die Arbeitsspeicher durch ein Bank-Switching Verfahren (nach dem Stand der Technik) derart zwischen den einzelnen Konfigurationen umzuschalten, daß die Debug-Information in einer jeweils neuen Bank liegt. Das Umschalten kann sehr zeitoptimierend, im Optimalfall sogar ohne Auswirkung auf die Verarbeitungsleistung erfolgen.

4. Funktionsweise des Debuggers

Das Debugger Programm selbst kann auf einem DB außerhalb des PA's ablaufen. Alternativ dazu kann eine VPU selbst den DB darstellen, entsprechend der Methoden, die bei Prozessoren angewendet werden. Dazu kann ein Task- oder Kontextswitch

(SWITCH) entsprechend den Ausführungen in PACT11 ausgeführt werden. Die Debuginformationen des zu debuggenden Programmes werden bei einem SWITCH zusammen mit den relevanten Daten gesichert und das Debugger Programm geladen, das die Informationen auswertet und/oder interaktiv mit dem Programmierer verarbeitet. Danach wird erneut ein SWITCH durchgeführt (bei welchem die relevanten Informationen des Debuggers gesichert werden) und das zu debuggende Programm wird weitergeführt.
Die Debug-Information wird von Debugger entsprechend Methode A und/oder B gelesen und in einen von der Datenverarbeitung separaten Speicher und/oder Speicherbereich gespeichert, auf den der DB bevorzugt direkten Zugriff besitzt.
Durch das Debugger-Programm werden die Breakpoints und (Vor)Bedingungen definiert. Ebenfalls kann das Debugger-Programm die Kontrolle über die Ausführung der Applikation, insbesondere den Ausführungsstart und das Ausführungende übernehmen.
Der Debugger stellt dem Programmierer eine geeignete Arbeitsumgebung zur Verfügung mit ggf. graphischer Oberfläche. In einer besonder bevorzugten Ausführung ist der Debugger in eine komplexe Entwicklungsumgebung integriert und tauscht mit dieser Daten und/oder Steuerinformation aus. Insbesondere kann der Debugger die aus den Arbeitsspeichern gelesenen Daten zur beliebigen Weiterverarbeitung auf einen Datenträger (Festplatte, CDROM) speichern und/oder innerhalb eines Netzwerkes (Ethernet) ablaufen.
Der erfindungsgemäße Debugger kann zudem gemäß PACT20 innerhalb einer Entwicklungsumgebung mit anderen Werkzeugen und insbesondere auch Debuggern kommunizieren; wobei in einer bevorzugten Ausgestaltung die Steuerung und/oder Definition der Debugparameter von einem anderen Debugger aus übernommen werden kann. Ebenso kann der Debugger die von ihm generierte De-

bug-Information einem anderen Debugger zur Verfügung stellen und/oder von diesem dessen Debug-Information erhalten. Insbesondere das Feststellen des Auftretens von Breakpoints und/oder (Vor)Bedingung ist durch einen anderen Debugger aus, bzw. den Einheiten die dieser andere Debugger debugt, durchführbar. Der erfindungsgemäße Debugger und die VPU reagieren dann entsprechend.
Der andere Debugger kann insbesondere der Debugger eines einer VPU zugeordneten anderen Prozessors sein.
Der andere Debugger kann insbesondere der Debugger eines mit einer VPU gekoppelten anderen Prozessors (CT, bzw. ARC bei Chameleon) sein.
Insbesondere kann der andere Debugger auf einem der VPU zugeordneten und/oder gekoppelten Prozessor ablaufen und/oder der zugeordnete Prozessor der DB sein, beispielsweise eine CT, bzw. ARC bei Chameleon. In einer besonders bevorzugten Ausgestaltung kann der zugeordnete Prozessor ein Host-Prozessor sein, wie beispielsweise in PACT26/US und/oder PACT28 beschrieben.

5. Bewertung der Methoden

Die Methode A ist erheblich zeit- und ressourcenintensiver als die Methode B, bei der kaum zusätzliche Hardware erforderlich ist und zudem ggf. das zeitaufwendige Auslesen der Debug-Information vom Start der Applikation an entfällt. Daher ist Methode B grundsätzlich vorzuziehen. Für Compiler nach PACT11 ist die Methode B eindeutig zu präferieren.
Es wurde erkannt, dass insbesondere die Verwendung beider Methoden A und B gemeinsam, zu den besten und transparentesten Debuggingergebnissen führt. Insbesondere kann je nach Tiefe des zu debuggenden Fehlers zunächst unter Zuhilfenahme der schnellen Debuggingmethode B debuggt werden und nach hinrei-

chende Lokalisierung des Fehlers sodann per Methode A die Details in der "Tiefe" analysiert werden.

6. Mixed Mode Debugger

Bei der Anwendung der besonders bevorzugten Methode B kann es zu dem Problem kommen, dass die in den Speichern befindliche sichtbare Information nicht hinreichend ist.

Typischerweise kann ein detaillierteres Debugging folgendermassen erfolgen:

a) Die sichtbare Debuginformation (PREINFO) vor der Konfiguration einer einen Breakpoint enthaltenden Konfiguration wird gesichert. Bei Auftreten eines Fehlers bei dem Breakpoint wird die dann sichtbare Debuginformation (POSTINFO) gesichert. Basierend auf der PREINFO Information wird ein Software-Simulator gestartetdie zu debuggende(n) Konfiguration(en), simuliert. Der Simulator kann dabei jeden Wert innerhalb der PAEs und der Bussysteme bestimmen und (ggf. auch graphisch und/oder textuell) ausgeben, wodurch ein detaillierter Einblick in den Ablauf des Algorithmus zum Zeitpunkt des Auftretens des Fehlers erreicht wird. Insbesondere ist es möglich die jeweils simulierten Werte mit den Werten von POSTINFO zu vergleichen um Differenzen schnell zu erkennen.

b) Die sichtbare Debuginformation vor einem Breakpoint wird gesichert. Bei Auftreten eines Breakpoints wird basierend auf dieser Information ein Software-Visualisierer gestartet. Der zu debuggende Baustein wird nunmehr in einem Single-Step Verfahren betrieben, das das Auslesen sämtlicher relevanter Daten nach Methode A ermöglicht. Diese Daten können nunmehr entweder direkt (ggf. auch graphisch und/oder textuell) ausgegeben werden und/oder an einen Simulator weitergeleitet werden, dessen Simulation sodann auf den detaillierten Daten beruht, und danach wie bekannt ausgegeben werden.

6.1 Vorteile eines Mixed Mode Debuggers

Der Mixed Mode Debugger ermöglicht die detaillierte Analyse der Abläufe innerhalb eines Bausteines. Durch die Möglichkeit gemäß Methode B bis zu einem gesetzten Breakpoints mit voller Geschwindigkeit zu arbeiten und danach ggf. anzuhalten und/oder ggf. in einen Single-Step Modus zu gehen, wird das Debugging zeiteffizient, wodurch das Testen großer Datenmengen und/oder komplexer Algorithmen ermöglicht wird.
Das bevorzugte Aufsetzen eines Simulators nach dem Auftreten des Breakpoints auf Basis der aktuellen Daten und Zustände ermöglicht einen genauen Einblick in die Hardware. Sofern der Zeitaufwand für die Simulation zu hoch ist und/oder die 100% Übereinstimmung des Simulators mit der Hardware fraglich ist, ermöglicht das Zurücklesen von Daten im Single-Step Modus nach Auftreten eines Breakpoints gemäß Methode A oder entsprechend des Kontext-Switch Verfahrens nach PACT15 und PACT11 ein 100% korrektes Debugging des Algorithmus und/oder auch der Hardware selbst.

7. Beschreibung der Figuren

Figur 1 und 2 entsprechen der Patentanmeldung PACT11. Die unterschiedlichen Ansätze der Methoden A und B wurden in die Figuren eingezeichnet (A, B)

Figur 1b zeigt eine Repräsentation des endlichen Automaten durch eine rekonfigurierbare Architektur nach PACT01 und PACT04 (PACT04 Fig. 12-15). Das kombinatorischen Netz aus Figur 1a (0101) wird durch eine Anordnung von PAEs 0107 ersetzt (0101b). Das Register (0102) wird durch einen Speicher

(0102b) ausgeführt, der mehrere Zyklen speichern kann. Die Rückkopplung gemäß 0105 erfolgt durch 0105b. Die Eingänge (0103b bzw. 0104b) sind äquivalent 0103 bzw 0104. Der direkte Zugriff auf 0102b kann ggf. durch einen Bus durch das Array 0101b realisiert werden. Der Ausgang 0106b ist wiederum äquivalent 0106.

Figur 2 zeigt die Abbildung eines endlichen Automaten auf eine rekonfigurierbare Architektur. 0201(x) repräsentieren das kombinatorische Netz (das entsprechend Figur 1b als PAEs ausgestaltet sein kann). Es existiert ein oder mehrere Speicher für Operanden (0202) und ein oder mehrere Speicher für Ergebnisse (0203). Zusätzlich Daten Ein-/Ausgänge gem. 0103b, 0104b, 0106b) sind der Einfachheit halber nicht dargestellt. Den Speichern zugeordnet ist jeweils ein Adressgenerator (0204, 0205).
Die Operanden- und Ergebnisspeicher (0202, 0203) sind physikalisch oder virtuell derart miteinander verkoppelt, daß beispielsweise die Ergebisse einer Funktion einer anderen als Operanden dienen können und/oder Ergebnisse und Operanden einer Funktion einer anderen als Operanden dienen können. Eine derartige Kopplung kann beispielsweise durch Bussysteme hergestellt werden oder durch eine (Re)Konfiguration durch welche die Funktion und Vernetzung der Speicher mit den 0201 neu konfiguriert wird.

Figur 3 zeigt eine mögliche schematische Struktur des Debuggings nach Methode B. Es sei insbesondere beispielhaft auf die Figuren 19, 20, 21 der Patentanmeldung PACT13 verwiesen, in welcher die Grundlage der Speicher beschrieben ist. PACT13

wir hiermit zu Offenbarungszwecken vollumfänglich eingegliedert.
0101b und 0102b ist wie bereits beschrieben dargestellt. Zusätzlich ist eine externer Speichereinheit dargestellt (0302), der möglicherweise ähnlich PACT13 an 0102b angeschlossen (0307) sein kann. Es soll besonders darauf hingewiesen werden, daß sowohl 0102b als auch 0302 externe oder interne Speichereinheiten sein können. Ebenfalls soll eine Speichereinheit als mindestes ein Register, eine Menge von Registern oder ein Speicher (RAM, Flash, o.ä.) oder Massenspeicher (Festplatte, Band, o.ä.) definiert sein.

Die debuggende Einheit 0301 kann Breakpoints innerhalb 0101b setzen (0303), auf Basis derer der eigentliche Debug Vorgang ausgelöst wird. Durch das Erreichen eines Breakpoints wird eine Information (0304) an 0301 gesendet, die den Debug Vorgang startet; zugleich werden auch alle 0101b interenen Vorkehrungen zum Debuggen (z.B. ein Anhalten und/oder Verlangsamen des Taktes) ausgelöst. Alternativ kann auch durch 0301 die Information generiert und an 0101b gesendet werden.
Über 0305 und/oder 0306 kann 0301 auf die Daten und/oder Zustände aus dem Speicher 0102b und/oder dem Speicher 0302 zugreifen. Der Zugriff kann zum Beispiel

1. durch eine Speicherkopplung (Block-Move, d.h. kopieren der Speicher in einen anderen, von 0301 kontrollierten Bereich)
2. durch eine Leitung (serielle oder parallele Leitung, über die ein oder mehrere Speicherbereich(e) übertragen wird/werden, z.B. JTAG)
3. Buskopplungen gleich welcher Art (die Speicher werden ähnlich eines DMA-Verfahren arbitriert und von 0301 verarbeitet)

erfolgen.

Als Beispiel wurde eine Figur aus PACT13 gewählt. Es soll ausdrücklich darauf hingewiesen werden, daß grundlegend jedes Speicherverfahren und jede Speicherkopplung (Stack, Random-Access, FIFO, usw.) entsprechend bearbeitet werden kann.

Die Figuren 4a und 4b zeigen weitere möglichen Ausgestaltungen und wurden aus der Patentanmeldung PACT28 entnommen, die zu Offenbarungszwecken vollumfänglich eingegliedert ist. Der Aufbau einer besonders bevorzugten VPU ist in Figur 4a dargestellt. Vorzugsweise hierarchische Konfigurationsmanager (CT's) (0401) steuern und verwalten eine Anordnung von rekonfigurierbaren Elementen (PACs) (0402). Den CT's ist ein lokaler Speicher für die Konfigurationen zugeordnet (0403). Der Speicher verfügt weiterhin über ein Interface (0404) zu einem globalen Speicher, der die Konfigurationsdaten zur Verfügung stellt. Über ein Interface (0405) sind die Konfigurationsabläuft steuerbar. Ein Interface der rekonfigurierbaren Elemente (0402) zur Ablaufsteuerung und Ereignisverwaltung (0406) ist vorhanden, ebenso ein Interface zum Datenaustausch (0407). Beipielsweise kann eine der CT's als DB agieren.

Figur 4b zeigt einen Ausschnitt aus einem beispielhaften CPU System, beispielsweise einem DSP des Types C6000 von Texas Instruments (0451). Dargestellt sind Programmspeicher (0452), Datenspeicher (0453), beliebige Peripherie (0454) und EMIF (0455). Über einen Speicherbus (0456) und einem Peripheriebus (0457) ist eine VPU als Coprozessor integriert (0458). Ein DMA-Kontroller (EDMA) (0459) kann beliebige DMA-Transfers, beispielsweise zwischen Speicher (0453) und VPU (0458) oder Speicher (0453) und Peripherie (0454) durchführen. In diesem Beispiel kann 0451 als DB arbeiten und insbesondere auch der

erfindungsgemäße Debugger mit dessem Debugger gekoppelt und/oder in diesen integriert sein.

Figur 5a zeigt einen beispielhaften Hardwareaufbau, der zum Debuggen von rekonfigurierbaren Prozessoren benutzt werden kann. Hierzu wird ein gepipelinter Konfigurationsbus 0501 verwendet, ähnlich dem aus PACT17 bekannten. Die Pipeline ist über mehrere Registerstufen (0502) in horizontaler und/oder vertikaler Richtung aufgebaut, um höhere Taktfrequenzen zu erreichen. Der gepipelinte Konfigurationsbus führt zu den zu konfigurierenden Elementen (PAEs) (0503), um diese mit Konfigurationsdaten zu versorgen.

Figur 5b zeigt beispielshaft die erforderliche Erweiterung gemäß der vorliegenden Erfindung. Jede Registerstufe (0502) dekrementiert den Zahlenwert (LATVAL) zum Ausgleich der Latenzzeit um 1 (angedeutet durch -1). Ebenso dekrementiert jede PAE (0503), die bereits eine Taktsteuerinformation erhalten hat, diese pro Takt um 1 (angedeutet durch -1/T). Auf die PAEs und insbesondere deren internen Register kann nunmehr nicht nur schreibend sondern auch lesend zugegriffen werden, z.B. über eine besondere Steuerleitung (RD), um Debugdaten auszulesen. Zu schreibende und gelesene Daten durchlaufen in diesem Beispiel das Bussystem durch das Array aus PAEs von links nach rechts und in der untersten Zeile in Rückwärtsrichtung. Der Konfigurationsbus ist weiterhin über Registerstufen (0505) pipelineartig zurückgeführt (0504). In diesem Beispiel kann eine übergeordnete Einheit (CT/Ladelogik, Hostprozessor) (0506) ebenso lesend und schreibend auf den Bus zugreifen, wie ein dediziertes Testinterface (0507). Das Testinterface kann einen eigenen Testkontroller aufweisen und insbesondere kompatibel zu einem oder mehrern marktgängigen Testschnittstellen sein (z.B. JTAG, Tektronix, Rhode&Schwarz,

etc). Die Auswahl der bussteuernden Einheit erfolgt über eine Multiplexer/Demultiplexer-Einheit (0508). In (0509 in Klammern und kursiv dargestellt) oder vor den Einheiten 0506 und 0507 kann eine Schaltung zur Rückrechung der Herkunftsadresse (0509) von Debugdaten, die über 0504 eintreffen, vorgesehen sein. Die Adressberechnungen innerhalb des aufgezeigten Systems geschehen wie folgt: Zunächst wird die Adresse durch 0506 oder 0507 am Bus 0501 angelegt. Die Adresse wird ähnlich der Verarbeitung der Zahlenwerte (LATVAL) zur Latenzberechnung in jeder Registerstufe (0502 **und** 0505) dekrementiert. Sobald die Adresse gleich 0 ist, wird die nach der Registerstufe liegende PAE selektiert. In der nachfolgenden Registerstufe wird die Adresse negativ, sodass keine weiteren PAEs mehr aktiviert werden. Sofern Daten aus einer PAE gelesen werden, werden diese zusammen mit der Adresse zurückübertragen. Die Adresse wird dabei in jeder Registerstufe weiter dekrementiert. Eine Rückrechnung in 0509 der bei 0506 und/oder 0507 zusammen mit den Debugdaten eintreffenden Adressen ist nunmehr durch eine einfache Addition möglich, indem die Anzahl der dekrementierenden Registerstufen zu dem eintreffenden Adresswert hinzuaddiert werden. Es soll angemerkt sein, dass die Registerstufen 0502 in Figur 5b leicht unterschiedlich gegenüber den Registerstufen 0502 in Figur 5a ausgestaltet sind. In Figur 5b weisen diese nämlich zusätzlich eine Schaltung (z.B. Multiplexer) zu Auswahl der weiterzuleitenden Daten auf, der entweder die Daten des Busses 0501 weiterschaltet oder den Ausgang der zugeordneten PAE (0503) und somit die DebugDaten. Zur Ansteuerung der Schaltung kann das Auftreffen des Adresswertes gleich 0 dienen.

Es wird nochmals darauf hingewiesen, dass das dedizierte Testinterface (0507) den Industriestandards entspricht. Es kann zu Tests während des Software-Debug-Vorgangs eingesetzt wer-

den und/oder zu Test während des Zusammenbaus von Hardwarekomponenten und -systemen (z.B. dem Zusammenbau der Schaltungen auf der Platine) und/oder zu Funktionstests des Halbleiterbausteins (Chips) im Rahmen der Halbleiterfertigung. Insbesondere dadurch kann die übliche Scan-Chain zum Test der Register während des Funktionstests des Halbleiters entfallen oder zumindest entsprechend minimiert werden, da dann nur die Register durch die Scan-Chain geführt werden müssen, die nicht vom Bussystem (0501) ansteuerbar sind.

Ebenfalls wird besonders darauf hingewiesen, dass das in Figur 5 erläuterte Verfahren keinesfalls auf die Anwendung von Konfigurationsbussen beschränkt ist. Gewöhnliche Datenbussysteme können ebenso zu den unterschiedlichen vorab aufgezählten Test- und Debugging-Zeitpunkten und -arten verwendet werden. Insbesondere sei in diesem Zusammenhang auf das in PACT07 beschriebene Datenbussystem hingewiesen. PACT07 ist hierzu zu Offenbarungszwecken vollumfänglich eingegliedert. Die in Figur 5 beschriebenen Verfahren können, für einen Ingenieur mit gewöhnlichen technischen Kenntnissen leicht nachvollziehbar ebenso auf PACT07 angewendet werden.
Ein Mischbetrieb unterschiedlicher Bussysteme, wie Konfigurationsbussystemen, Datenbussystemen nach PACT07 und gewöhnlichen Datenbussystemen ist desweiteren grundsätzlich möglich. Dazu können mehrere Testinterface vorgesehen sein, oder wie technisch bevorzugt die Multiplexer/Demultiplexerstufe (0508) auf eine Mehrzahl von Bussystemen (n x 0501, n x 0504) ausgelegt werden.

Abschließend soll noch besonders erwähnt sein, dass durch die Rückführung des Bussystems nach Figur 5b auch die in die PAEs zu schreibenden Konfigurationsdaten zurückgeführt werden. Unter Zuhilfenahme der Adressrückrechung (0509) und der zurück-

geführten Statusleitung REJ, die nach PACT17, PACT10, PACT05 eine Zurückweisung der Konfigurationsdaten anzeigt, kann auf die Verwendung der Konfigurationszwischenspeicher-FIFOs nach PACT17 Figuren 8 und 9 (0805, 0903) verzichtet werden, da deren Funktionalität nunmehr vollumfänglich über das beschriebene Bussystem abgebildet wird.

8. Begriffsdefinition

lokal relevanter Zustand Zustand der nur innerhalb einer bestimmten Konfiguration relevant ist

global relevanter Zustand Zustand der in mehreren Konfigurationen relevant ist und zwischen den Konfigurationen ausgetauscht werden muß

relevanter Zustand Zustand der innerhalb eines Algorithmus zur korrekten Ausführung dessen benötigt wird und somit durch den Algorithmus beschrieben ist und verwendet wird

irrelevanter Zustand Zustand der für den eigentlichen Algorithmus ohne Bedeutung ist und auch nicht im Algorithmus beschrieben ist, der jedoch von der ausführenden Hardware implementierungsabhängig benötigt wird

Akte: PACT21d

Anmelder: Pact XPP Technologies AG
Muthmannstr. 1
80939 München

Vertreter: Patentanwalt
Claus Peter Pietruk
Heinrich-Lilienfein-Weg 5
D-76229 Karlsruhe
Vertreter-Nr. 321 605

Deutsche Patentanmeldung

Titel: Verfahren zum Debuggen rekonfigurierbarer Architekturen

Patentansprüche

Verfahren zum Debuggen von rekonfigurierbarer Hardware, dadurch gekennzeichnet, daß sämtliche notwendige Debuginformation je Konfigurationszyklus in einen Speicher geschrieben werden, der dann vom Debugger ausgewertet wird.

-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-

Fig. 1b

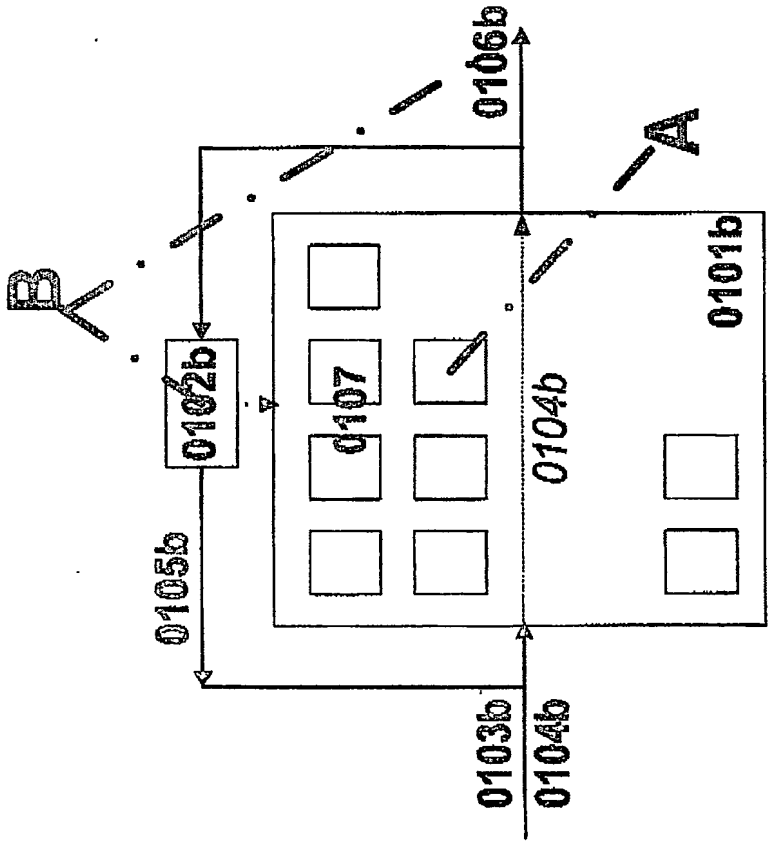
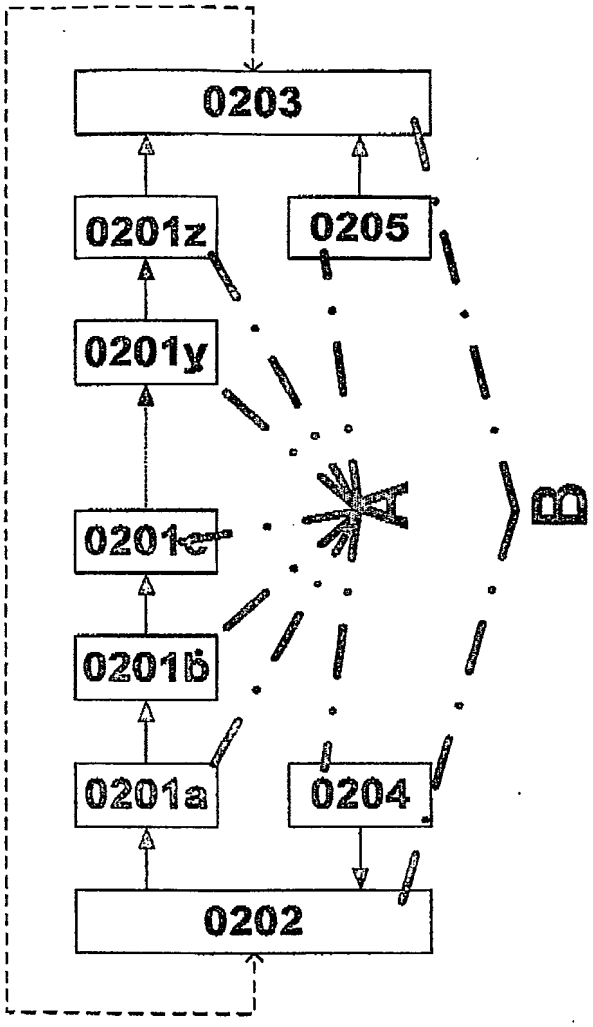


Fig. 2



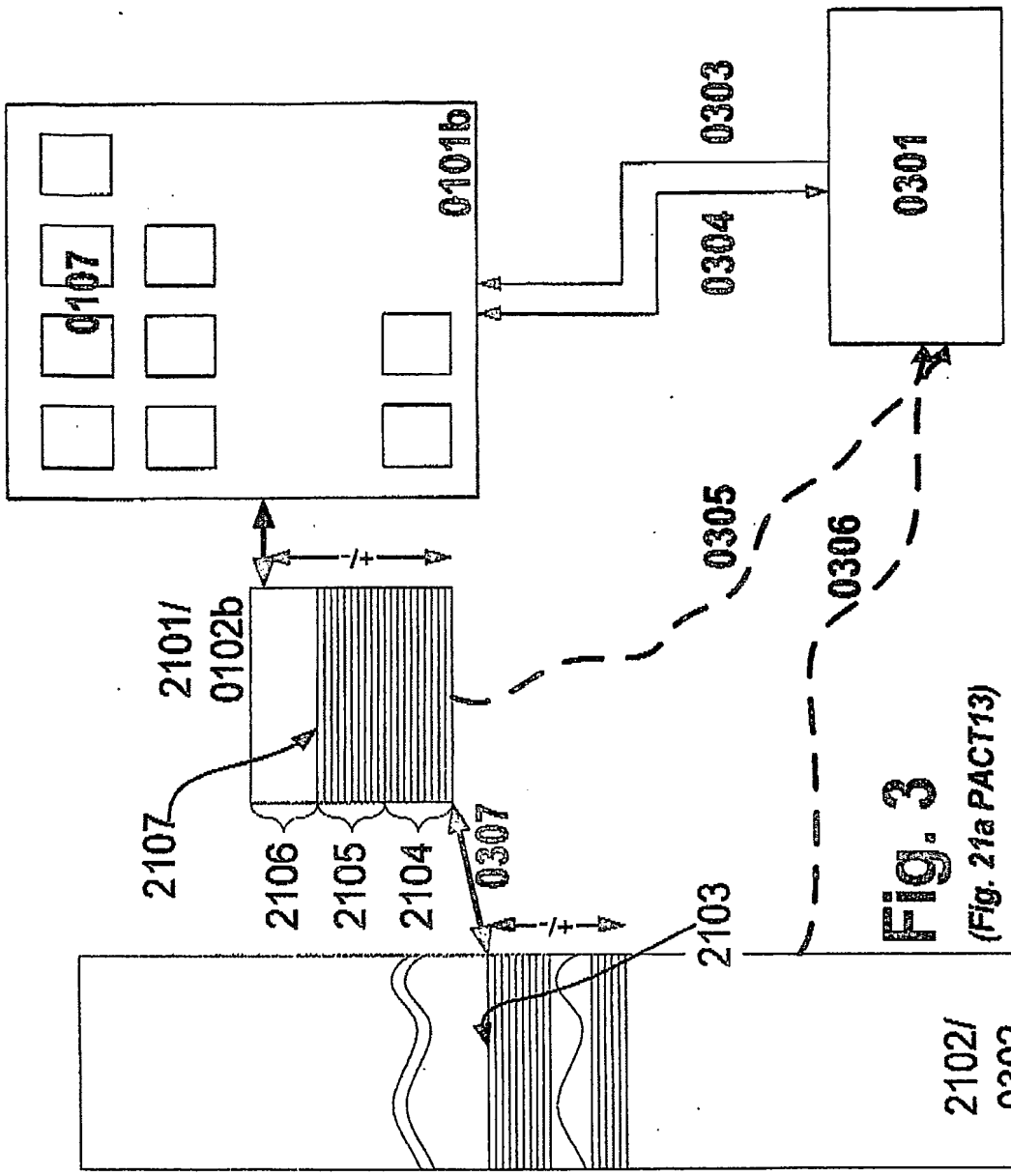


FIG. 3
(Fig. 21a PACT13)

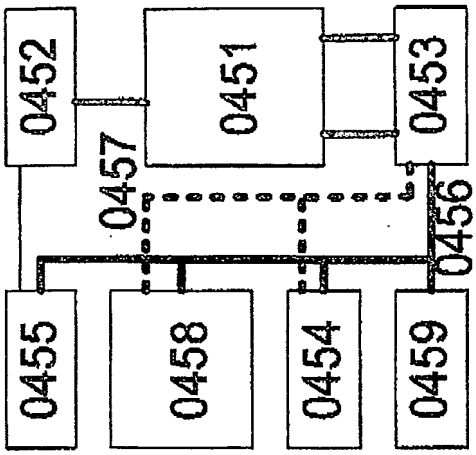

0452
0451
0453
0457
0456
0455
0458
0454
0459

Fig. 4b

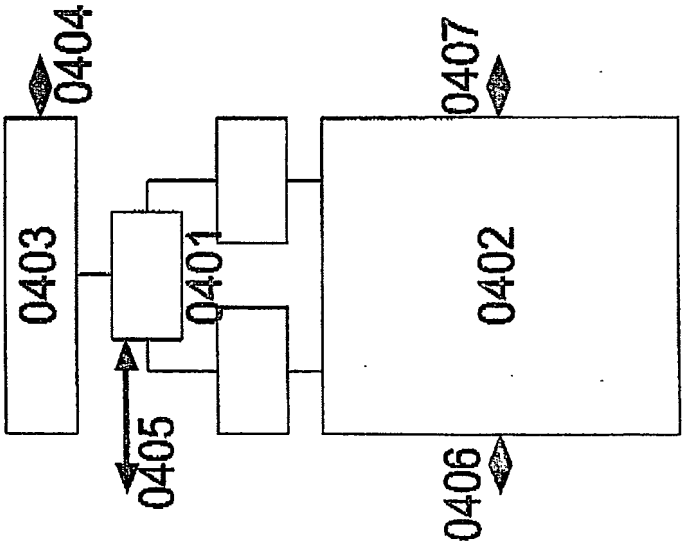

0404
0407
0403
0401
0402
0405
0406

Fig. 4a

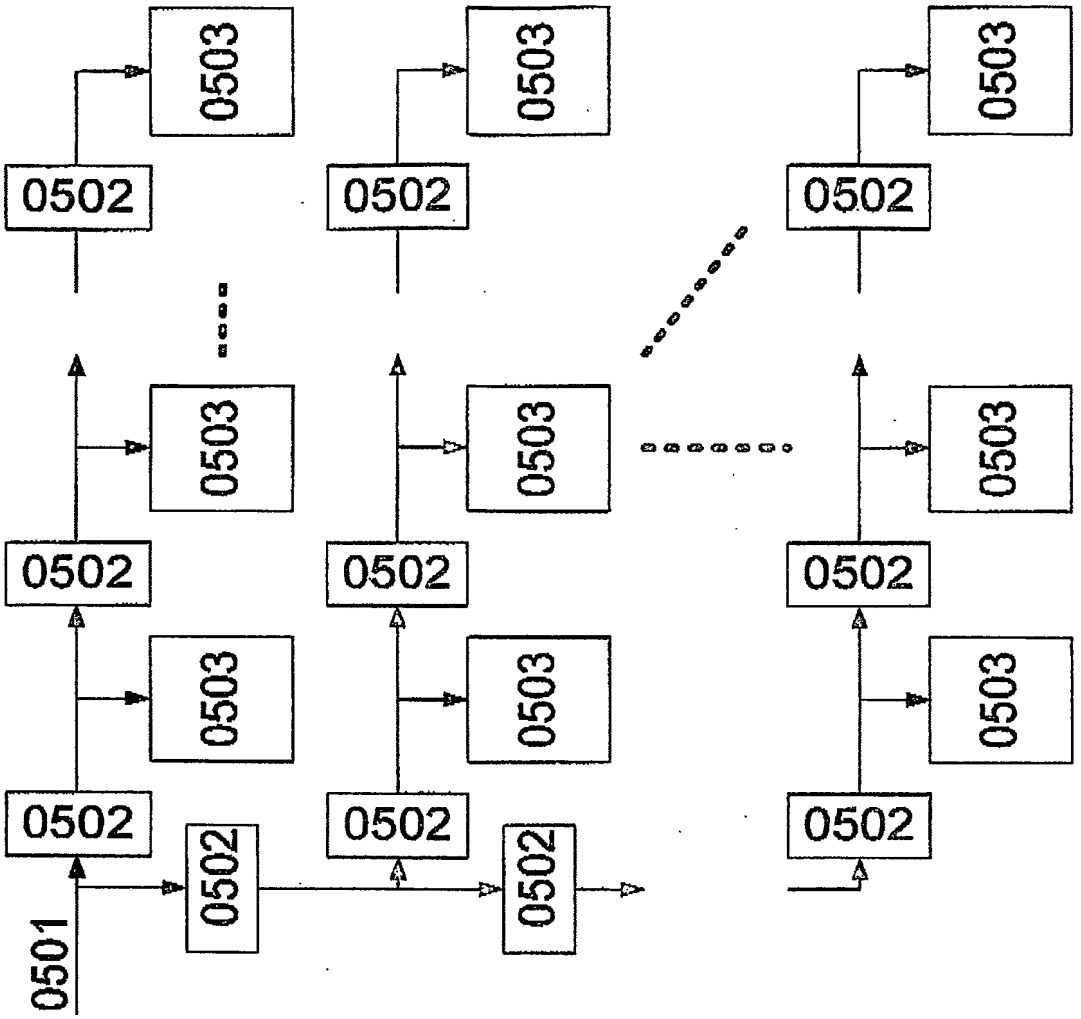


Fig. 5a

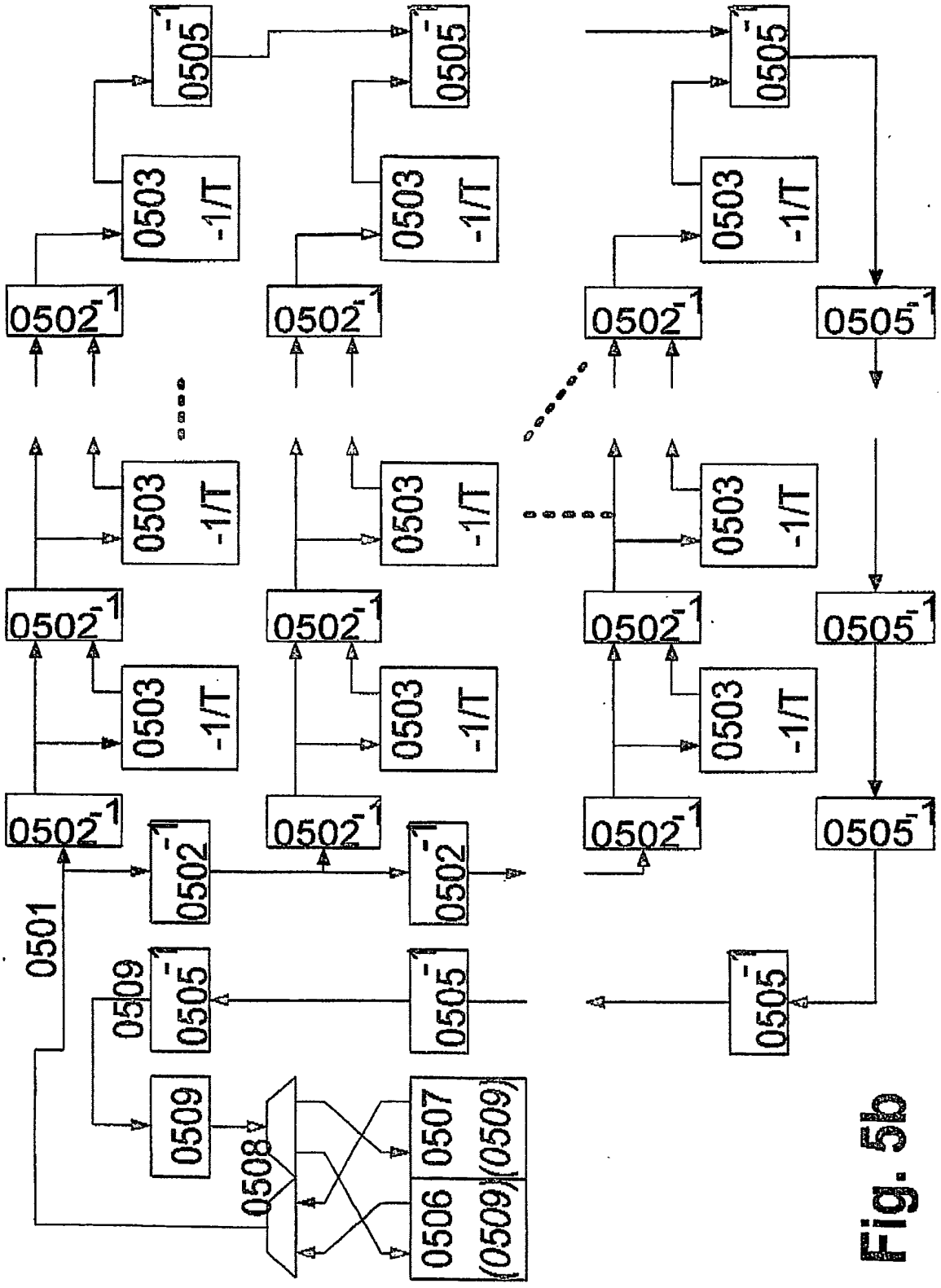


Fig. 5b

GESAMT SEITEN 36